
Vers une meilleure compréhension de la différence théorique entre un composant et un service

Anthony Hock-koon, Mourad Oussalah

Laboratoire LINA (CNRS - UMR 6241)
Université de Nantes - Bât 11
2 rue de la Houssinière – B.P. 92208
44322 Nantes Cedex 03
anthony.hock-koon@univ-nantes.fr
mourad.oussalah@univ-nantes.fr

RÉSUMÉ. Une distinction claire et suffisamment consensuelle entre objet et composant fut longue à se dessiner. Ainsi l'apparition du paradigme service a amené naturellement la communauté de l'ingénierie logicielle à s'interroger sur ses limites. En effet, service et composant partagent de nombreux points communs et la frontière entre leurs spécificités et leurs apports respectifs restent assez flou. Dans cet article nous essayons de faire un premier travail de séparation en proposant un cadre de comparaison basé sur un triptyque produit, processus et qualité.

ABSTRACT. The distinction between Objects and Components is well-known, but the distinction between Components and Services is much less clear. Therefore, we define the Product-Process-Quality framework which intends to provide a clear understanding of the differences between Component-based software engineering (CBSE) and Service-Oriented software engineering (SOSE). Moreover, this framework handles the Object orientation (OO) to provide a global point of view of these three styles of software engineering.

MOTS-CLÉS : Objet, Composant, Service, Cadre de Comparaison Conceptuel

KEYWORDS: Object, Component, Service, Conceptual Comparison Framework

1. Introduction

L'ingénierie logicielle à base de composants (Component-based software engineering - CBSE [HEI 01]) et l'ingénierie logicielle orientée services (Service-oriented software engineering - SOSE [STO 05, THE 08]) sont deux domaines établis du génie logiciel. Le CBSE s'inspire des autres disciplines de l'ingénierie telles que l'ingénierie mécanique ou électrique. Il repose sur l'idée de "construire des systèmes à partir de composants". Le SOSE se base sur le modèle de mondialisation des entreprises modernes. Le principe est de sous-traiter les fonctions à faible valeur ajoutée ou échappant au domaine de compétence de l'entreprise afin de se concentrer sur les aspects innovants du système.

Ainsi CBSE et SOSE ont la réutilisation d'entités logicielles existantes, composants ou services, comme socle commun. Tous deux reposent sur le concept d'architecture logicielle [MED 00] dans lequel un système est vu comme une structure claire d'entités et de relations entre ces entités. Le SOSE est un paradigme plus récent dont l'approche est influencée par le CBSE d'une façon similaire à l'incidence que l'objet a eu sur le composant. De plus, ces dernières années ont vu CBSE et SOSE coexister tout en ayant un développement parallèle ce qui a entraîné l'apparition de concepts similaires ou spécialisés qui sont souvent mis en juxtaposition avec des confusions de vocabulaire. Les applications modernes qui combinent composants et services ajoutent à cette confusion ambiante et la compréhension globale devient plus difficile.

Notre objectif est de clarifier les frontières entre CBSE et SOSE. Dans cet article, nous proposons un premier cadre de comparaison basé sur un triptyque produit, processus et qualité. Ce triptyque organise les caractéristiques de chacun afin d'offrir aux architectes une meilleure compréhension des implications et des conséquences du choix de l'un ou l'autre des paradigmes. De plus, ce cadre de comparaison prend en compte l'orienté objet (OO) afin d'apporter une vision globale de l'évolution des préoccupations du génie logiciel entre objet, composant et service.

La section suivante va présenter les différences théoriques que nous identifions entre CBSE et SOSE. Dans la section 3 nous présentons l'aspect quantitatif de notre cadre de comparaison qui regroupe les notions de produit et de processus. La section 4 va en aborder l'aspect qualitatif. La section 5 discute des contributions de notre travail et les limitations des autres tentatives de comparaisons existantes. Enfin la section 6 conclut cet article.

2. Différences théoriques entre CBSE et SOSE

CBSE et SOSE ont une approche très similaire et partagent les activités d'identification des entités logicielles (composant ou service) qui répondent aux besoins, puis de combinaison de ces entités pour réaliser l'application globale. Ils reposent sur les mêmes notions de composition et de composite qui garantissent une approche homogène où toute entité est un composant ou un service. Cependant, bien que CBSE et

SOSE aient le même objectif global, les théories derrière les composants et les services sont différentes.

Un composant est dit off-the-shelf (sur l'étagère) [NIT 08, CRN 06, HEI 01] par adoption d'une *pièce* technologique, le composant, qui est disponible pour les développeurs. Un service est centré sur l'utilisation d'une *fonction* fournie par un tiers [STO 05, OAS 08, THE 08]. Ces deux visions sont très proches et notre travail veut raffiner leur définition.

Pour illustrer notre première distinction nous prenons un exemple de l'industrie du jeu vidéo sur PC. Cette industrie repose sur deux modèles de distribution de contenus. Le premier modèle classique correspond à un joueur qui achète une copie de son jeu. Le joueur est ensuite responsable du déploiement de cette copie sur sa propre machine. Ce modèle correspond à une approche composant. Typiquement, le jeu (composant) vient avec un manuel d'instructions (documentation) sur la configuration système minimale requise pour être capable d'exécuter ce jeu ; puis sur les opérations nécessaires pour l'installer ; et enfin sur comment y jouer. Tous ces éléments définissent des contraintes du côté client. Le second modèle appelé Cloud gaming illustre la notion de service. Le joueur paie le droit de jouer à un jeu qui est exécuté sur une plate-forme à distance sous la responsabilité du fournisseur. Il a uniquement besoin d'une interface et d'une connexion pour accéder à cette plate-forme. Il n'est plus responsable du déploiement du jeu sur sa machine. Les seules informations qui lui sont nécessaires sont : comment accéder à cette plate-forme et comment jouer au jeu. Ainsi, le joueur n'a plus à s'occuper des contraintes du système pour exécuter le jeu qui conserve une qualité constante. Au contraire du premier modèle où il peut être amené à améliorer la configuration de sa machine pour obtenir une certaine expérience du jeu qui va varier d'un système à un autre.

Ainsi, l'orienté service pousse la responsabilité du propriétaire à son extrême. En effet, l'approche off-the-shelf de CBSE implique que seul le développement, la qualité de service et la maintenance sont sous la responsabilité du fournisseur, tandis qu'en SOSE le fournisseur est aussi responsable du déploiement, de l'exécution et la gestion du service. La localisation physique du service et ses évolutions sont transparentes pour l'utilisateur.

Ce concept de responsabilité implique aussi la nature *multitenant* du service, c'est-à-dire que le service exécuté à distance doit être capable de gérer de multiples connexions parallèles. Ce principe n'est pas nécessaire à un composant. En effet, bien que pouvant appartenir à de multiples compositions, au niveau de l'exécution, différentes instances du composant sont créées et chacune d'elles sous la responsabilité d'un client.

Ainsi, CBSE et SOSE ont des points de vue différents sur les relations entre client et fournisseur qui agissent sur d'autres aspects de distinction : la gestion des hétérogénéités, l'importance de l'automatisation et le couplage.

L'objectif du service est l'indépendance avec les technologies d'implémentation. Un service doit être accessible et utilisable sans aucune hypothèses sur son implémen-

tation, sur les utilisateurs potentiels ou sur la façon d'utiliser ce service. Cette problématique est bien connue en CBSE mais ne représente pas un enjeu aussi critique. En effet, il existe un très grand nombre de modèles de composant [CRN 07]. Un architecte doit choisir un modèle particulier et n'utiliser que des composants respectant ce modèle car la collaboration entre modèles différents est très difficile [CRN 06] (par exemple les ponts entre COM et CORBA). Le SOSE prône un unique modèle homogène de services [ERI 08] qui doit être standardisé et utilisé par tous.

Une autre distinction est l'importance accordée à l'automatisation qui a participé à la définition même du SOSE. En effet, la grande majorité des travaux cherche à automatiser les mécanismes du SOSE tels que la publication de services, les découvertes et sélections, la composition, etc. Ce principe d'automatisation est poussé à son extrême par le concept d'auto-adaptation [NIT 08] qui cherche à coordonner l'ensemble des mécanismes liés au service afin d'assurer des adaptations contextuelles réactives ou proactives. Bien que l'automatisation est un élément clé de la recherche en CBSE elle ne fait pas partie de la définition d'un modèle de composant [CRN 07].

Gestion des hétérogénéités et automatisation participent à une autre distinction entre CBSE et SOSE : le couplage [HOC 10a]. Le SOSE prône un couplage faible à tous les niveaux, du développement à l'exécution. D'une part, un architecte ne doit pas connaître, au moment de la spécification de son application, les services qui seront réellement utilisés. Ce couplage faible entre l'expression des besoins et les services concrets disponibles est supporté par les principes de découvertes et sélections dynamiques. D'autre part, le SOSE va chercher à réduire au maximum les dépendances entre services concrets en collaboration pour faciliter l'auto-adaptation [NIT 08].

Toutes les distinctions que nous avons identifiées sont liées à la notion de dynamique. Le SOSE vient de besoins fonctionnels d'applications spécifiques en terme d'exigences sur l'adaptabilité et l'agilité alors que le CBSE a une vocation plus générale.

Dans [CRN 06], les auteurs définissent le CBSE par les mots clés *composabilité* et *réutilisabilité*. Nous définissons le SOSE par la *composabilité*, la *réutilisabilité* et la *dynamicité*.

3. Les aspects quantitatifs

Cette section classe les éléments structurels et les mécanismes qui caractérisent le CBSE et le SOSE. Nous ne cherchons pas à être exhaustifs mais plutôt à mettre en évidence leurs différences par une liste pertinente des concepts centraux. Ces concepts sont répertoriés dans les catégories produits et processus.

3.1. Produits et processus

Un **produit** est une entité logicielle ou conceptuelle qui est le résultat d'une action ou d'un processus. Un **processus** est une action ou une succession d'actions qui est utilisée pour créer ou modifier un produit et ainsi obtenir un produit en résultat. Les produits sont répartis en deux sous catégories :

- **éléments architecturaux simples** : les briques élémentaires de construction d'un paradigme ;
- **éléments architecturaux composites** : les produits complexes construits à partir d'éléments architecturaux existants. Leur structure identifie clairement les éléments architecturaux réutilisés et leurs relations.

Chaque sous catégorie est à son tour divisée en deux groupes suivant les deux *niveaux d'abstraction* : le *design-time* et le *runtime*.

La catégorie des processus se focalise sur le principe de réutilisation, c'est-à-dire comment réutiliser des entités logicielles, les *constituants*, pour en construire de nouvelles, les *composites* (un constituant pouvant être un élément architectural simple ou composite). Ces notions de constituants et composites définissent deux *niveaux de description*. Ainsi les processus sont regroupés suivant les niveaux d'abstraction et de description.

- **Dans un niveau d'abstraction** - regroupe les processus qui ciblent et produisent des produits d'un même niveau d'abstraction (Figure 1 flèches blanches). Cette catégorie est divisée en design-time et runtime ;
- **Entre niveaux de description** - regroupe les processus qui ciblent des produits de deux niveaux de description différents (Figure 1 flèches hachurées). Cette catégorie est divisée en design-time et runtime ;
- **Entre niveaux d'abstraction** - regroupe les processus qui assurent le passage des produits entre design-time et runtime (Figure 1 flèches noires).
- **processus de cycle de vie** : regroupe les processus chargés des différentes activités du cycle de vie, de l'idée de la conception d'une application, à son développement et son utilisation et jusqu'à sa fin de vie. Cette sous catégorie est divisée en deux suivant les notions de *for reuse* et *by reuse* liées à la construction en constituants et composites. La catégorie *for reuse* se focalise sur le processus de développement d'un constituant dans le but d'être réutilisé. La catégorie *by reuse* se focalise sur le processus de développement d'un composite par réutilisation de constituants existants.

La figure 1 montre la répartition des produits et processus sur un exemple. Un composite C est construit à partir de deux constituants : un élément architectural simple (constituant A) et un élément architectural composite (constituant B). Ces trois produits ont leurs représentations au design-time et au runtime. Les flèches représentent les processus qui sont étudiés. Les flèches blanches sont les processus liés à un même niveau d'abstraction, design-time ou runtime. Les flèches hachurées sont les processus qui font le lien entre niveaux de description et qui ont leurs représentations au design-

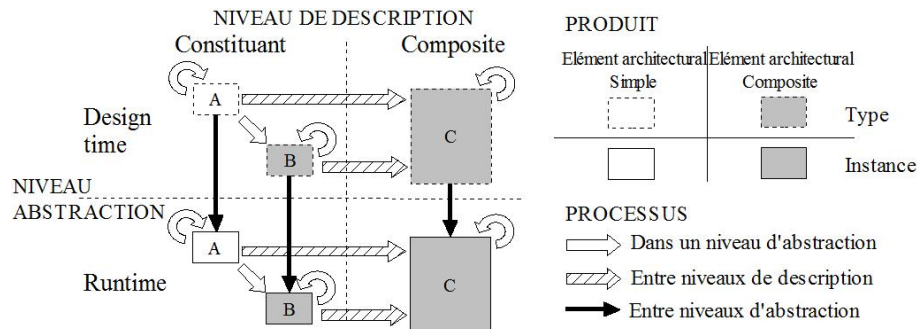


Figure 1. Niveaux d'abstraction et de description

time et au runtime. Les flèches noires sont les processus qui font le lien entre niveaux d'abstraction.

3.2. Comparaison entre objet, composant et service

3.2.1. Produit

Éléments architecturaux simples - (Tableau 1) Les éléments architecturaux simples de l'orienté objet sont la *classe* au design-time et son instance, l'*objet*, au runtime. La même distinction est faite pour le CBSE entre les produits *type de composant* et *type de connecteur* [GAR 97], et leurs instances *composant* et *connecteur*. En SOSE, la frontière entre les niveaux d'abstraction est beaucoup moins claire et la plupart des travaux existants se réfèrent à un *service* comme une entité du runtime [STO 05, THE 08]. Chaque service est associé à une *description de service* [OAS 08] qui est la cible des nombreux processus runtime.

Éléments architecturaux complexes - (Tableau 1) Les trois paradigmes partagent la notion de *composite*. L'orienté objet repose sur les notions de *classe composite* et d'*objet composite*. Le CBSE repose au design-time sur les notions de *type de configuration* et *type de composant composite*, et au runtime sur leurs instances *configurations* et *composants composites*. Le SOSE est principalement au runtime. Ainsi une composition de services dépend des notions de *schémas de composition* et de *service composite*. Ces notions sont associées à deux patrons de coordination de services, la *chorégraphie* et l'*orchestration* [OAS 08]. Ces patrons sont typiquement produits et exécutés au runtime.

3.2.2. Processus

L'objectif de cette partie n'est pas de fournir une liste exhaustive des processus existants mais de sélectionner les plus pertinents et largement acceptés afin de mettre en perspective les différences entre objet, composant et service.

Dans un niveau d'abstraction - (Tableau 1) Le SOSE étant principalement défini au runtime la première sous catégorie suivante ne le concerne pas.

- Design-time : l'orienté objet repose principalement sur les processus d'*association* et d'*héritage*. Le CBSE repose sur la *composition horizontale* entre éléments architecturaux de même niveau de description. Mais on peut également citer les processus de *versionnement* ou de *raffinage*.

- Runtime : les processus de communication entre éléments architecturaux sont la préoccupation principale de cette catégorie. OO et CBSE reposent principalement sur différents processus d'*appels* alors que le SOSE inclut obligatoirement des processus additionnels. Typiquement, les services constituants doivent être découverts et sélectionnés dynamiquement (processus de *service provisioning*). Ensuite ces services constituants coordonnent leurs actions par un processus de *chorégraphie* qui définit les successions d'*invocations de service*. De plus, un processus de *publication de services* est nécessaire pour mettre un service à disposition des clients potentiels.

Entre niveaux de description - (Tableau 1)

- Design-time : l'OO repose sur le processus de *composition* pour produire des éléments composites. Le CBSE repose sur la *composition verticale* qui fait le lien entre constituants et composites.

- Runtime : les processus de communication entre constituants et composites sont l'essence de cette catégorie. OO et CBSE reposent sur différents processus d'appels. En CBSE, ces appels sont parfois nommés comme le processus de *délégation*. En SOSE, la coordination des processus d'*invocations de services* entre composite et constituants est appelée l'*orchestration*.

Entre niveaux d'abstraction - (Tableau 1) Objet et composant reposent sur le processus d'*instanciation* pour lier type et instance. Le SOSE se focalisant sur le runtime cette catégorie ne le concerne pas.

Processus de cycle de vie - la principale différence avec l'OO vient d'une séparation entre le développement *for reuse* et *by reuse*. Le *for reuse* est le processus de développement d'un composant ou d'un service en vue d'être réutilisé. Le *by reuse* est le processus de développement d'un système en réutilisant des composants ou des services existants. CBSE et SOSE vont se différencier principalement sur la notion de découverte et sélection dynamique de services et sur l'absence de déploiement des services constituants sélectionnés. Un certain nombre d'activités du cycle de vie sont modifiées ainsi que la frontière entre le design-time et runtime. Nous ne détaillons pas cet aspect dans l'article due à la limitation du nombre de pages.

3.3. Analyse des aspects quantitatifs

Le tableau 1 résume la partie quantitative de notre cadre de comparaison. Comme nous l'avons remarqué précédemment les définitions couramment admises du SOSE se focalisent sur la partie runtime. Cependant, certains travaux [CAV 09, HOC 10a]

Tableau 1. Produit-Processus : Comparaison Objet, Composant et Service

Produit		OBJET	COMPOSANT	SERVICE
Élément architectural simple	Design-time	Classe	Type composant, Type connecteur	(Service abstrait)
	Runtime	Objet	Composant, Connecteur	Service (Concret), Description
Élément architectural composite	Design-time	Classe composite	Type de configuration, Type composant composite	(Schéma de composition abstrait), (Service composite abstrait)
	Runtime	Objet composite	Configuration, Composant composite	Schéma de composition (Concret), Patron de chorégraphie/orchestration, Service composite (Concret), (Description service composite)
Dans un niveau d'abstraction	Design-time	Association, Héritage	Composition horizontale, Versionnement, Raffinage	(Choreograph), (Héritage de schéma de composition)
	Runtime	Appel de méthode	Appel de fonction	Chorégraphie, Provisioning, Invocation, Publication
Entre niveaux de description	Design-time	Composition	Composition verticale	(Orchestration), (Description de Composition de services)
	Runtime	Appel de méthode	Appel de fonction, Délégation	Orchestration, Invocation
Entre niveaux abstraction		Instanciation	Instanciation	(Provisioning), (Instanciation de schéma de composition)
Processus de cycle de vie	For reuse		Processus de développement de composant	Processus de développement de service
	By reuse	Cycle de vie de système basé objets	Cycle de vie de système basé composants	Cycle de vie de système basé services

proposent une analyse plus fine des différents produits et processus afin d'améliorer le degré d'expressivité du SOSE en gérant de façon explicite les deux niveaux d'abstraction. Ces propositions sont représentées dans le tableau 1 par les éléments en *(italic)*.

Dans [CAV 09], les auteurs introduisent le concept de *service abstrait*. Le processus de service provisioning utilise ce service abstrait pour sélectionner le *service concret* qui correspond aux besoins. Le service concret est le service que le système va réellement utiliser et invoquer au runtime. Cependant, les services abstraits et concrets sont définis comme des instances de services ce qui implique une confusion entre les niveaux d'abstraction et les notions de types et instances. Ainsi, nous proposons [HOC 10a] une autre approche du service abstrait qui est vu comme un type de service. Le processus qui fait le lien entre le type et l'instance est le service provisioning. Ainsi la méthode d'instanciation par provisioning du SOSE est différente des approches OO et CBSE. Le *service abstrait* est un produit du design-time. Le *service concret* un produit du runtime. Le *service provisioning* est un processus de la catégorie du design-time au runtime.

D'une façon homogène nous proposons les concepts de *schéma de composition abstrait* et de *service composite abstrait* et leur instance respective *schéma de composition concret* et *service composite concret*. Le *service provisioning* est à nouveau le processus qui fait le lien entre service composite abstrait et concret cependant nous introduisons un processus d'*instanciation de schémas de composition*. En effet, un service composite repose sur un schéma de composition (orchestration ou chorégraphie) pour coordonner ces services constituants. Ce composite est multitenant et doit donc gérer de nombreux clients en parallèle. Ainsi, à partir du schéma de composition abstrait qui exprime le comportement du service composite, le processus d'instanciation va créer un schéma de composition concret pour chaque client et ainsi isoler les exécutions.

De plus, l'introduction de la notion de schéma de composition abstrait et concret va permettre la définition de processus inspirés de l'objet et du composant. Un processus d'*héritage* sur ce schéma peut faciliter la réutilisation. Le fournisseur de service composite pourra facilement développer des comportements spécialisés définis après négociation avec certains clients. Le service composite maintiendra les différents schémas de composition abstraits et pourra les instancier de façon adéquate en fonction des clients qui se connectent.

La partie quantitative de notre cadre de comparaison met aussi en évidence une autre lacune du SOSE sur la notion de *description de service composite*. La description de service est un élément clé de la dynamique en service cependant il n'existe pas encore d'étude approfondie sur la définition d'une description de service composite et sur un processus de *composition de description de services* qui permettrait d'automatiser la mise à disposition d'un composite sur le marché.

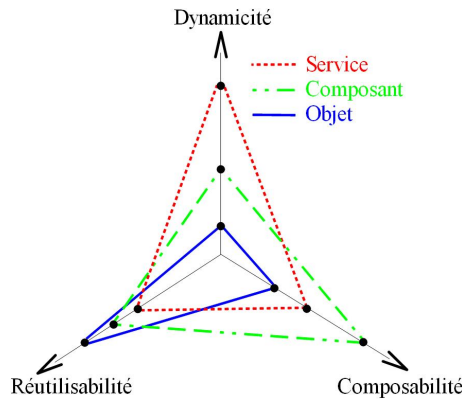


Figure 2. *Qualité globale*

4. Les aspects qualitatifs

Les travaux existants sur la qualité logicielle introduisent un grand nombre de critères basés sur différents points de vue (développeur, utilisateur, etc.). Notre analyse qualitative ne cherche pas à faire une liste exhaustive de ces critères mais plutôt à extraire l'essence des paradigmes de développement logiciel afin de pouvoir les comparer. Ainsi, notre approche peut se diviser en trois étapes. Dans un premier temps, nous cherchons à abstraire les qualités primordiales d'un paradigme. Puis nous identifions les principales caractéristiques d'un paradigme qui impactent sur ces qualités. Enfin, ces caractéristiques sont utilisées pour classer OO, CBSE et SOSE et les comparer en fonction des qualités.

4.1. Critères de qualité

Notre première contribution décrite dans la section 2 identifie nos qualités primordiales : réutilisabilité, composabilité et dynamique.

- **Réutilisabilité** : la probabilité qu'un produit ou un processus soit réutilisé avec peu ou pas de modifications ;
- **Composabilité** : la facilité d'un paradigme de développement logiciel à combiner de façon sûre ses éléments architecturaux dans le but de construire de nouveaux systèmes ou éléments composites.
- **Dynamicité** : se base sur la facilité d'un paradigme de développement logiciel à construire et modifier une application et le niveau d'automatisation lié aux processus associés.

Ces trois critères ont des impacts intuitivement compréhensibles sur des propriétés telles que la flexibilité, l'agilité, la productivité, la robustesse, l'extensibilité, etc.

La figure 2 montre le positionnement suivant nos critères des paradigmes objet, composant et service qui sont à l'origine de leur définition. La réutilisation est l'enjeu central de l'OO mais il ne propose que très peu de solutions pour améliorer la composabilité et la dynamique. En effet, ces préoccupations ne sont devenues capitales que bien plus tard, en particulier la composabilité a poussé l'apparition du composant. Ainsi, le CBSE a apporté un grand nombre d'innovations et permet de poser les bases d'une dynamique accrue. Cependant son approche trop générique pose des contraintes sur ces solutions qu'une partie de la communauté a voulu contourner en proposant la notion de service. Ainsi, le développement du SOSE est exclusivement centré sur la dynamique mais repose sur le socle posé par les autres paradigmes. Dans cette section nous cherchons à raffiner cette vision résumée très haut niveau.

Nous identifions six caractéristiques principales des paradigmes de développement logiciel qui impactent sur la réutilisabilité, la composabilité et la dynamique.

Couplage Faible - le couplage mesure les dépendances entre entités en collaboration et affaiblir ce couplage s'est tendre vers leur indépendance. Dans [HOC 10a] nous avons étudié la notion de couplage faible que nous définissons comme une combinaison de trois couplages distincts : sémantique, syntaxique et physique. Le couplage sémantique se focalise sur l'expertise du designer sur le domaine d'application de son système. Il lui permet d'en établir la criticité des aspects fonctionnels. Le couplage syntaxique étudie les relations entre ces aspects fonctionnels et leurs représentations physiques dans le système. Enfin, le couplage physique mesure les dépendances entre ces représentations physiques. Cette définition du couplage se base sur de nombreux éléments tels que la gestion des hétérogénéités, les processus de communication, la complexité des données échangées, etc.

Pouvoir expressif - représente la capacité du paradigme à proposer un grand nombre de concepts et de processus permettant de spécifier et de manipuler ses produits.

Abstraction de communication - la capacité du paradigme à abstraire la couche de communication qui dirige l'exécution de l'application.

Architecture explicite - la capacité du paradigme à fournir une vue architecturale claire de l'application.

Pouvoir évolutif - la capacité du paradigme à proposer un ensemble de concepts et de processus pour faire évoluer ses éléments architecturaux.

Responsabilité propriétaire - correspond à la répartition des responsabilités entre fournisseur et consommateur d'un bloc logiciel réutilisé (objet, composant, service). Ces responsabilités sont : le développement, la qualité de service, la maintenance, le déploiement, l'exécution et la gestion. Ce critère exprime le degré de liberté accordé au client par le fournisseur sur son produit.

Ces six caractéristiques ont des impacts forts et entremêlés sur nos trois critères de qualité, cependant nous cherchons toujours à qualifier l'essentiel de chaque élément afin de fournir un cadre de comparaison simple mais explicite. Ainsi, la figure 3 montre

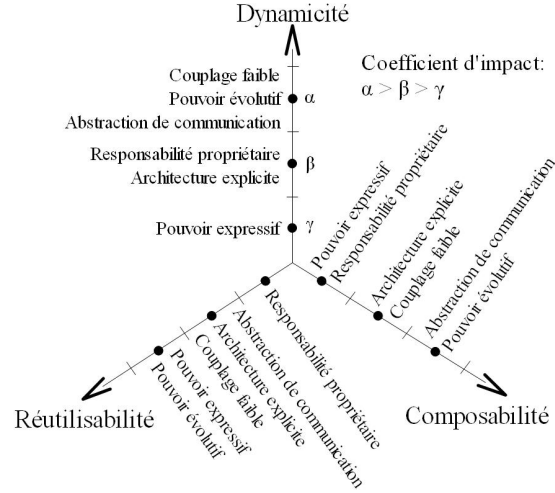


Figure 3. *Hiérarchie caractéristique et qualité*

les relations entre les caractéristiques et les critères de qualité identifiés. Ces caractéristiques sont organisées du groupe γ qui a l'impact le moins important au groupe α qui l'impact le plus important. À partir de cette classification nous définissons un ensemble de formule.

$$\begin{aligned} Dynamicite = & \alpha * (Cplg + Evol + AbstrCom) + \beta * (Resp + ArchiExpl) \\ & + \gamma * (Expr) \end{aligned} \quad [1]$$

$$\begin{aligned} Composabilite = & \alpha * (Evol + AbstrCom) + \beta * (Cplg + ArchiExpl) \\ & + \gamma * (Resp + Expr) \end{aligned} \quad [2]$$

$$\begin{aligned} Reutilisabilite = & \alpha * (Evol + Expr) + \beta * (AbstrCom + ArchiExpl + Cplg) \\ & + \gamma * (Resp) \end{aligned} \quad [3]$$

$$\alpha > \beta > \gamma$$

Nous ne détaillons pas la définition de cette organisation hiérarchique afin de nous focaliser sur son utilisation pour la comparaison entre objet, composant et service.

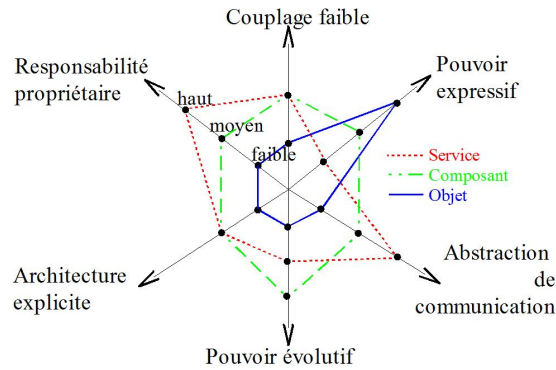


Figure 4. *Comparaison objet, composant et service*

4.2. Comparaison entre objet, composant et service

La figure 4 montre notre classification entre OO, CBSE et SOSE en fonction de nos six caractéristiques suivant trois niveaux d'importance (haut, moyen, faible) accordés sur chacun d'eux.

Couplage Faible - typiquement, les systèmes à base d'objets font intervenir un ensemble de classes fortement couplées tandis que CBSE et SOSE ciblent une réduction de ce couplage. Le couplage faible est un enjeu clé de ces deux paradigmes. L'entrelacement des solutions proposées et leurs influences mutuelles ne permettent pas de les différencier de manière claire. Cependant, notre étude du couplage [HOC 10a] permet d'identifier une marge de progression possible.

Pouvoir expressif - l'OO manipule un grand nombre de concepts tels que l'héritage, les niveaux d'abstraction, les niveaux de description, la granularité, la réflexion, etc. Le CBSE s'est largement inspiré de ce travail mais il n'a pas encore atteint le même niveau de maturité sur les concepts comme l'héritage, la réflexion, etc. Enfin, le SOSE possède le plus faible pouvoir expressif car il combine les mêmes lacunes que le composant auxquelles s'ajoutent des imprécisions sur les niveaux d'abstraction soulignés par notre analyse quantitative.

Abstraction de communication - le SOSE fournit la meilleure abstraction de communication basée sur l'encapsulation apportée par les services et l'isolation des communications dans un schéma de composition. Le comportement global est plus facilement localisable, compréhensible et manipulable. De plus, cet aspect est renforcé par la nature multitenant du service et sa capacité à gérer des exécutions parallèles. En CBSE, les communications sont localisées à l'intérieur des différents connecteurs qui partagent le comportement global. Les flots de travail sont moins explicites. La granularité très fine du OO accentue cet handicap.

Architecture explicite - contrairement à l'OO, CBSE et SOSE reposent directement sur le concept d'architecture logicielle.

Pouvoir évolutif - le pouvoir évolutif est directement lié à la notion d'architecture explicite. Une architecture logicielle définit clairement des entités et les relations entre ces entités. On peut la représenter par un graphe basé sur des noeuds et des arcs. Les processus d'évolutions peuvent être regroupés suivant leur cible, le noeud, l'arc ou le graphe. Typiquement, l'OO ne propose pas cette notion d'architecture explicite et donc de graphe. Les processus d'évolution de l'OO se focalisent uniquement sur les noeuds et les arcs. Au contraire, CBSE et SOSE manipulent une architecture explicite et donc offrent des processus d'évolutions sur les trois cibles. Cependant, la maturité plus importante du CBSE et sa gestion explicite des niveaux d'abstraction ont permis à sa communauté d'aller encore plus loin et de proposer des processus d'évolutions aux niveaux meta et meta meta architecture [GOA 08, GAR 09].

Responsabilité propriétaire - le SOSE pousse la responsabilité propriétaire à son extrême où le fournisseur de service est responsable du développement, de la qualité de service, de la maintenance, du déploiement, de l'exécution et de la gestion. Au contraire, le CBSE partage les responsabilités au niveau du déploiement où le client devient responsable de l'instanciation du composant dans son application, de son exécution et de sa gestion. En OO, la classe est en boîte blanche où le client est libre de la manipuler à sa guise.

À partir de cette classification (Figure 4), nous utilisons les formules d'évaluation précédentes pour mesurer la réutilisabilité, la composabilité et la dynamique de chaque paradigme. Chaque niveau d'importance (haut, moyen, faible) est associé à une valeur (resp. 3, 2, 1).

$$\begin{aligned}
 \text{Objet : } Dynamicite &= \alpha(3) + \beta(2) + \gamma(3) \\
 Composabilite &= \alpha(2) + \beta(2) + \gamma(4) \\
 Reutilisabilite &= \alpha(4) + \beta(3) + \gamma(1)
 \end{aligned} \tag{4}$$

$$\begin{aligned}
 \text{Composant : } Dynamicite &= \alpha(7) + \beta(4) + \gamma(2) \\
 Composabilite &= \alpha(5) + \beta(4) + \gamma(4) \\
 Reutilisabilite &= \alpha(5) + \beta(6) + \gamma(2)
 \end{aligned} \tag{5}$$

$$\begin{aligned}
 \text{Service : } Dynamicite &= \alpha(7) + \beta(5) + \gamma(1) \\
 Composabilite &= \alpha(5) + \beta(4) + \gamma(4) \\
 Reutilisabilite &= \alpha(3) + \beta(7) + \gamma(3)
 \end{aligned} \tag{6}$$

Ainsi, on peut remarquer que CBSE et SOSE sont très proches avec une meilleure dynamique pour le service face à une meilleure réutilisation pour le composant. Cette similarité illustre l'origine de la confusion entre ces deux paradigmes. Cependant, cet article met en évidence les différences sur les causes qui amènent ce résultat.

5. Contributions

Malgré son importance, peu de travaux se focalisent sur la distinction entre composants et services. Nous identifions une seule autre étude de cette comparaison. Dans [BRE 07] les auteurs listent les concepts et les vocabulaires liés aux notions de composant et service et cherchent à expliciter les termes ambigus. Notre approche est différente mais complémentaire. En effet, leur travail a une approche plus fine sur les aspects produits mais aborde très peu les aspects processus et qualité. D'autres comparaisons peuvent être trouvées [ATT 09, NIT 08] et qui nous ont aidé à forger notre démarche mais elles n'abordent cette problématique que de façon parcellaire. Cet article veut être un complément aux recherches existantes sur la différence entre objet et composant [SZY 02, OUS 05] en y incluant le service. Nous proposons donc un cadre de comparaison global de trois paradigmes principaux du génie logiciel.

L'aspect quantitatif a mis en évidence certaines carences du SOSE sur la gestion des niveaux d'abstraction dont nous avons fourni quelque pistes de solutions. L'aspect qualitatif apporte une meilleure compréhension des principes de réutilisabilité, composabilité et dynamique en soulignant les caractéristiques fondamentales de chacun d'eux et peut ainsi servir de guide aux architectes. Par exemple, dans [HOC 10b] nous cherchons à définir un méta modèle de service composite qui supporte les compositions dynamiques. Notre service composite repose sur une architecture claire qui identifie les rôles de chacun des services constituants et les communications. De plus, il utilise des mécanismes de médiation pour réduire le couplage. Ainsi, notre méta modèle se focalise sur trois caractéristiques qui se trouvent au plus haut niveau de la dynamique et de la composabilité (composition dynamique) : le couplage faible, l'architecture explicite, l'abstraction de communication (Figure 3).

6. Conclusion

Dans cet article nous proposons une compréhension claire des différences théoriques entre les notions de composant et de service. Ce travail repose sur un cadre de comparaison basé sur un triptyque produit, processus et qualité. De plus, notre comparaison inclut l'orienté objet afin d'offrir un point de vue global sur l'évolution des préoccupations du génie logiciel.

7. Bibliographie

- [ATT 09] ATTIOGBÉ C., « Can Component/Service-Based Systems Be Proved Correct ? », *SOFSEM*, 2009, p. 3-18.
- [BRE 07] BREIVOLD H. P., LARSSON M., « Component-Based and Service-Oriented Software Engineering : Key Concepts and Principles », *EUROMICRO-SEAA*, 2007, p. 13-20.
- [CAV 09] CAVALLARO L., NITTO E. D., PRADELLA M., « An Automatic Approach to Enable Replacement of Conversational Services », *ICSOC/ServiceWave*, 2009, p. 159-174.

- [CRN 06] CRNKOVIC I., CHAUDRON M. R. V., LARSSON S., « Component-Based Development Process and Component Lifecycle », *ICSEA*, 2006, page 44.
- [CRN 07] CRNKOVIC I., CHAUDRON M., SENTILLES S., VULGARAKIS A., « A Classification Framework for Component Models », *Proceedings of the 7th Conference on Software Engineering and Practice in Sweden*, 2007.
- [ERI 08] ERICKSON J., SIAU K., « Web Services, Service-Oriented Computing, and Service-Oriented Architecture : Separating Hype from Reality », *J. Database Manag.*, vol. 19, n° 3, 2008, p. 42-54.
- [GAR 97] GARLAN D., MONROE R. T., WILE D., « Acme : an architecture description interchange language », *CASCON*, 1997, page 7.
- [GAR 09] GARLAN D., BARNES J. M., SCHMERL B. R., CELIKU O., « Evolution styles : Foundations and tool support for software architecture evolution », *WICSA/ECSA*, 2009, p. 131-140.
- [GOA 08] GOAER O. L., TAMZALIT D., OUSSALAH M., SERIAI A., « Evolution Shelf : Reusing Evolution Expertise within Component-Based Software Architectures », *COMPSAC*, 2008, p. 311-318.
- [HEI 01] HEINEMAN G., COUNCILL W., *Component based software engineering : putting the pieces together*, Addison-Wesley Professional, 2001.
- [HOC 10a] HOCK-KOON A., OUSSALAH M., « Defining Metrics for Loose Coupling Evaluation in Service Composition », *IEEE SCC*, 2010, p. 362-369.
- [HOC 10b] HOCK-KOON A., OUSSALAH M., « Expliciting a Composite Service by a Meta-Modeling Approach », *RCIS*, 2010.
- [MED 00] MEDVIDOVIC N., TAYLOR R. N., « A Classification and Comparison Framework for Software Architecture Description Languages », *IEEE Trans. Software Eng.*, vol. 26, n° 1, 2000, p. 70-93.
- [NIT 08] NITTO E. D., GHEZZI C., METZGER A., PAPAZOGLU M. P., POHL K., « A journey to highly dynamic, self-adaptive service-based applications », *Autom. Softw. Eng.*, vol. 15, n° 3-4, 2008, p. 313-341.
- [OAS 08] OASIS, « Reference Architecture for Service Oriented Architecture 1.0 », , 2008, <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf>.
- [OUS 05] OUSSALAH M., ALL, *Ingénierie des composants : Concepts, techniques et Outils*, Vuibert, 2005.
- [STO 05] STOJANOVIC Z., DAHANAYAKE A., *Service-oriented Software System Engineering Challenges And Practices*, IGI Publishing, Hershey, PA, USA, 2005.
- [SZY 02] SZYBERSKI C., *Component Software : Beyond Object-Oriented Programming*, Addison-Wesley Professional, 2002.
- [THE 08] THESECSETEAM, « Service Centric System Engineering », *EU Integrated Project*, , 2008, <http://www.secse-project.eu/>.